Universidade Federal de Santa Catarina – UFSC
Programa de Pós-Graduação em Ciência da Computação
INE 410130 – Computação Distribuída
Semestre 2023/1

Definição do Projeto Final

Você deverá desenvolver uma biblioteca de comunicação confiável, capaz de garantir entregas de mensagens para um grupo de processos respeitando critérios de ordem de entrega causal e total. Dessa forma, implementações de algoritmos ou sistemas que utilizem a biblioteca poderão usufruir dessas garantias no recebimento de mensagens.

A biblioteca deve disponibilizar para o programador as primitivas *send*(*id, m*) e *receive*(*m*) para mensagens *unicast,* destinadas a um destinatário (comunicação 1:1), onde *id* é o identificador do destinatário e *m* é uma mensagem; e primitivas *broadcast*(*m*) e *deliver*(*m*) para mensagens destinadas a todos os participantes (comunicação 1:*n*), sendo *m* uma mensagem e *n* o número total de participantes.

Ordem de entrega: A entrega de mensagens *unicast* (1:1) deverá respeitar a ordem de entrega causal e a entrega de mensagens *broadcast* (1:*n*) respeitará a ordem de entrega total.

Por razão de simplificação, você pode considerar que a mensagem *m* é composta por um array de bytes. A serialização/deserialização das mensagens, considerando que o usuário queira trocar dados de tipos diferentes de byte array fica por conta do usuário (a biblioteca não precisa disponibilizar um mecanismo próprio para serialização de mensagens para outros tipos de dados).

- *Ordem causal*: Para garantias de entrega respeitando a ordem causal, você pode seguir a implementação do algoritmo proposto em (Raynal et al., 1991).
- *Ordem total*: Para implementação de broadcast com entrega em ordem total, você pode se basear em qualquer uma das propostas apresentadas em (Defago et al., 2004).

Grupo de processos: As garantias de ordem devem ser respeitadas para um conjunto de processos participantes na comunicação. Observe que a ordem causal implementa, internamente, um vetor de relógios lógicos de *n* posições, onde *n* representa o número de processos. Para a ordenação total, também são garantidas a entrega de mensagens ordenadas para *n* processos participantes da comunicação. Portanto, a biblioteca levará em consideração um grupo de processos comunicantes definidos antecipadamente. Por exemplo, as informações sobre os *n* processos podem ser definidas em arquivos de configuração ou informadas pela linha de comando na inicialização dos processos. Assuma que o número de processos participantes não muda durante a execução do programa. O quadro a seguir ilustra uma possível identificação dos processos em um arquivo de configuração.

```
//config_nodo0          //config_nodo1          //config_nodo2
processos = 3           processos = 3           processos = 3
id = 0                  id = 1                  id = 2
0 = localhost:6000      0 = localhost:6000      0 = localhost:6000
1 = localhost:6001      1 = localhost:6001      1 = localhost:6001
2 = localhost:6002      2 = localhost:6002      2 = localhost:6002
```

Detalhes de Implementação: A biblioteca deve implementar a comunicação distribuída entre os participantes usando sockets ou algum outro mecanismo de comunicação remota ponto a ponto. Não há restrições sobre a linguagem de programação utilizada.

Para simular atrasos na entrega de mensagens e assegurar que os critérios de ordenação estão sendo respeitados, você pode adicionar um tempo de espera aleatório (um *sleep*) durante a execução das primitivas *receive(m)* e *deliver*(*m*).

Serviço de Replicação: Para testar a biblioteca, serão implementados dois serviços de replicação, conforme a descrição abaixo:

- Serviço de replicação passiva sem líder fixo: Este serviço é composto por **n** réplicas, sendo que os clientes se comunicam diretamente com qualquer uma das réplicas. As réplicas, por sua vez, ao receberem uma requisição de um cliente, repassam a requisição às demais réplicas. Tanto na comunicação entre cliente e réplica ou entre uma réplica e as demais réplicas, é utilizada a primitiva *unicast* com entrega causal. A Figura 1 ilustra um cenário com dois clientes enviando requisições (mensagens m1 e m2) para o conjunto de réplicas. Na ilustração, o cliente C1 escolheu a réplica R2 para comunicar e R2 repassou a requisição às demais réplicas. Analogamente, o cliente Ck escolheu a réplica R1, que repassou a mensagem m1 às demais réplicas.

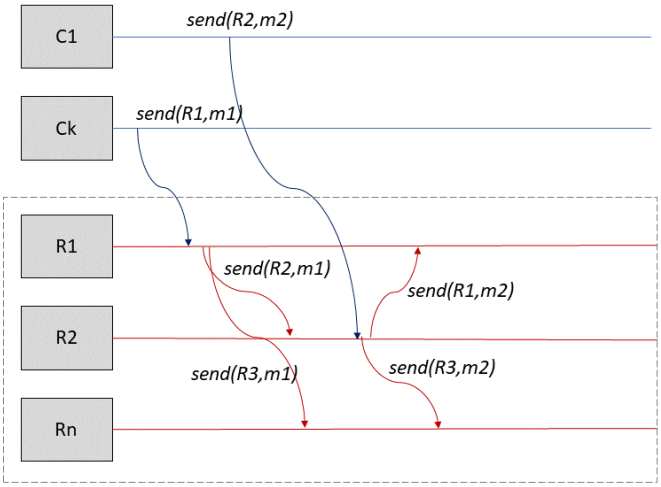
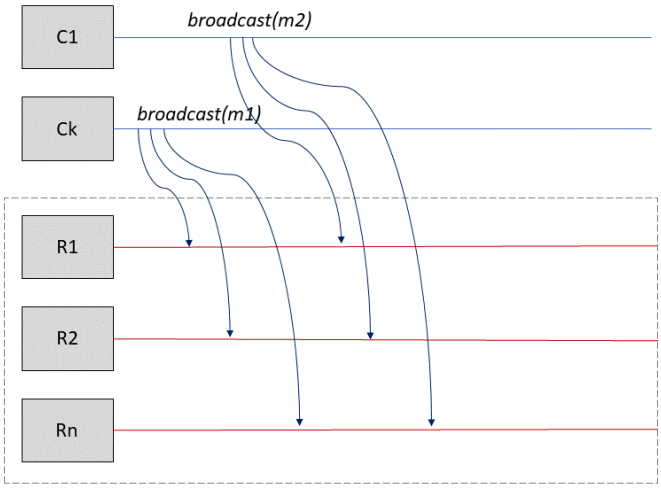


Figura 1: Serviço de replicação com unicast

- Replicação Ativa: No modelo de replicação ativa, cada requisição é enviada a todas as réplicas por difusão com garantia de ordem total. A Figura 2 ilustra o cenário com o cliente C1 enviando a mensagem m2 com a primitiva *broadcast(m2)*. Analogamente, o cliente Ck envia a mensagem m1 às réplicas com a primitiva *broadcast(m1)*.



Não há nenhuma imposição ao serviço que será implementado pelas réplicas. Apenas como exemplos, pode-se implementar um serviço de armazenamento do tipo chave-valor, um servidor de travas distribuído, um serviço de *logging*, entre outros. É importante apenas que seja possível identificar quais mensagens chegaram, e em que ordem, em cada réplica. Por exemplo, cada réplica pode imprimir na tela ou em um arquivo a sequência de requisições recebidas.

Avaliação

A data e horário da apresentação do projeto devem ser **agendados previamente** por e-mail ou mensagem Moodle, preferencialmente em horário de aula ou no horário de atendimento do professor, com antecedência mínima de 24 horas. O prazo final para apresentação é dia **29/06/2023**.

Na apresentação deve ser executada a implementação. Na sequência, deve ser mostrado o código do programa e respondidas as perguntas sobre o código feitas pelo professor.

O código-fonte deve ser enviado ao professor para análise e avaliação.

O projeto deve ser desenvolvido individualmente.

Em caso de cópia do projeto de qualquer fonte, todos os alunos envolvidos terão nota igual a zero no projeto.

Referências

Defago, X., Schiper, A., Urban, P., Défago, X., Schiper, A., Urban, P., & Urbán, P. (2004). Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey. *ACM Computing Surveys*, *36*(4), 372–421. https://doi.org/10.1145/1041680.1041682

Raynal, M., Schiper, A., & Toueg, S. (1991). The causal ordering abstraction and a simple way to implement it. *Information Processing Letters*, *39*(6), 343–350.